TÓM LƯỢC BÀI GIẢNG

(Vũ Quốc Hoàng)

ĐỆ QUI TRÊN DANH SÁCH LIÊN KẾT

Chủ đề

- Đệ qui trên danh sách liên kết

Tài liệu

[1] Tony Gaddis, *Starting out with C++ From Control Structures through Objects*, Pearson, 8th edition, 2015.

[2] Vũ Quốc Hoàng, *Bí kíp luyện Lập trình C (Quyển 1)*, hBook, 2017.

Đọc tài liệu

- Đệ qui trên danh sách liên kết: Chương 19 [1], Bài 4.6 [2].

Kiến thức

- Ta đã biết nhiều ưu điểm của việc dùng danh sách liên kết thay vì mảng để hiện thực danh sách. Tuy nhiên, một nhược điểm quan trọng của danh sách liên kết (so với mảng) là danh sách liên kết không có khả năng truy cập phần tử tùy ý (random access) như mảng, ta phải *truy cập tuần tự* (sequential access) các phần tử trên danh sách liên kết. Các thao tác trên danh sách liên kết được cài đặt hiệu quả bằng kĩ thuật lặp, nhưng ta cũng có thể cài đặt chúng bằng kĩ thuật đệ qui. Vì cách truy cập tuần tự như đã nói nên việc đệ qui trên danh sách liên kết không linh động như trên mảng mà hạn chế như trên chuỗi của C.
- Giả sử cấu trúc nút của danh sách liên kết đơn (singly linked list) được khai báo như sau:

```
struct NODE
{
    int data;
    NODE *next;
};
```

Hàm sau minh họa việc tính chiều dài danh sách bằng kĩ thuật đệ qui, mà "cơ bản", nó hoàn toàn giống việc tính chiều dài chuỗi trên kiểu chuỗi của C.

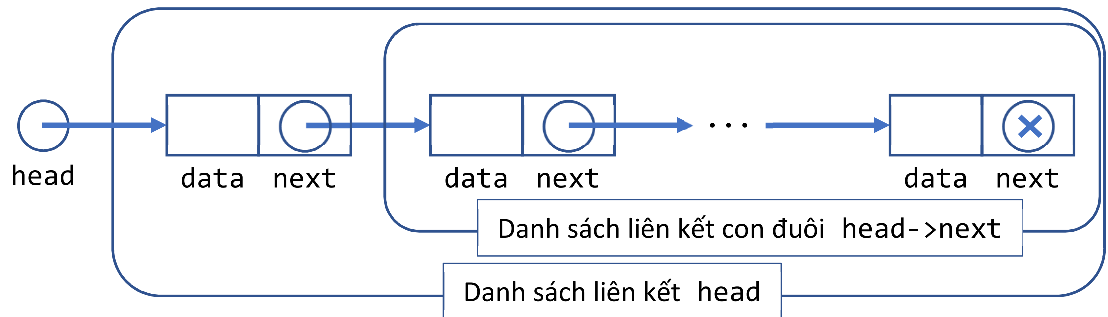
```
// Hàm tính chiều dài DSLK
int length(NODE *head)
{
    if(head == NULL)
        return 0;

    return 1 + length(head->next);
}
```

```
// Hàm tính chiều dài chuỗi
int length(const char *str)
{
    if(*str == '\0')
        return 0;

    return 1 + length(str + 1);
}
```

- Trường hợp cơ sở là trường hợp danh sách liên kết rỗng (head == NULL), khi đó, nó có chiều dài là 0. Trường hợp danh sách liên kết không rỗng thì chiều dài của nó là chiều dài của “danh sách liên kết con đuôi” cộng thêm 1 như hình minh họa sau.



- Ta đã biết về bài toán tìm kiếm trên danh sách và biết rằng tìm kiếm nhị phân thì hiệu quả hơn rất nhiều so với tìm kiếm tuần tự. Tuy nhiên với hạn chế truy cập tuần tự, ta không thể tìm kiếm nhị phân trên danh sách liên kết như trên mảng, mà <u>chỉ có thể tìm kiếm tuần tự trên danh sách liên kết</u>. Hàm sau minh họa việc tìm kiếm tuần tự trên danh sách liên kết bằng kĩ thuật đệ qui, mà “cơ bản”, nó hoàn toàn giống hàm đệ qui trên.

```
NODE* linear_search(NODE *head, int key)
{
    if(head == NULL || head->data == key)
        return head;

    return linear_search(head->next, key);
}
```

- Trường hợp cơ sở là trường hợp danh sách liên kết rỗng, khi đó, hàm trả về con trỏ NULL để “báo là không tìm thấy”. Trường hợp cơ sở thứ 2 là trường hợp nút đầu tiên của danh sách liên kết chứa phần tử thỏa tiêu chí tìm kiếm, khi đó, hàm trả về con trỏ đến nút này (head). Trường hợp còn lại, một cách đệ qui, hàm trả về con trỏ đến nút chứa phần tử cần tìm trên “danh sách liên kết con đuôi”.
- Ta cũng đã biết về bài toán sắp xếp một danh sách và biết các thuật toán sắp xếp đơn giản là sắp xếp chèn (Insertion Sort) và sắp xếp chọn (Insertion Sort). Hơn nữa, ta cũng đã biết cách cài đặt đệ qui các thuật toán này trên mảng. Ở đây, ta sẽ thực hiện một thuật toán sắp xếp hiệu quả hơn nhiều là Quicksort. Quicksort cũng có thể được cài đặt đệ qui trên mảng nhưng việc cài đặt đệ qui trên danh sách liên kết lại tỏ ra tự nhiên và đơn giản hơn. Để sắp tăng dần danh sách L, ý tưởng cơ bản của Quicksort đơn giản như sau:
    - *Trường hợp cơ sở*: nếu L rỗng hoặc chỉ có một phần tử, ta không cần làm gì cả
    - *Trường hợp đệ qui*: gọi $pivot$ là phần tử đầu của L, duyệt tuần tự qua các phần tử của L và so giá trị với $pivot$, ta tách L thành 3 danh sách con như sau:
        - Danh sách a gồm các phần tử nhỏ hơn $pivot$
        - Danh sách m gồm các phần tử bằng $pivot$
        - Danh sách b gồm các phần tử lớn hơn $pivot$

Sau đó, bằng đệ qui, ta sắp xếp tăng dần các danh sách a và b, rồi nối (theo trình tự) danh sách a (đã sắp) với danh sách m với danh sách b (đã sắp) để được danh sách sắp tăng dần của L.

Hàm đệ qui sau hiện thực ý tưởng trên.

```
void quicksort(NODE *&head, NODE *&tail)
{
    if(head == tail) // trường hợp cơ sở
        return;

    // chia (tách danh sách con a, b, m)
    int pivot = head->data;
    NODE *m_head = head, *m_tail = head;
    NODE *a_head = NULL, *a_tail = NULL;
    NODE *b_head = NULL, *b_tail = NULL;
    for(NODE *p = head->next; p != NULL; p = p->next)
    {
        if(p->data == pivot) // thêm vào danh sách m
        {
            m_tail->next = p;
            m_tail = p;
        }
        else if(p->data < pivot) // thêm vào danh sách a
        {
            if(a_tail == NULL)
                a_head = a_tail = p;
            else
            {
                a_tail->next = p;
                a_tail = p;
            }
        }
        else // thêm vào danh sách b
        {
            if(b_tail == NULL)
                b_head = b_tail = p;
            else
            {
                b_tail->next = p;
                b_tail = p;
            }
        }
    }
    if(a_tail != NULL)
        a_tail->next = NULL;
    if(b_tail != NULL)
        b_tail->next = NULL;

    // trị bằng đệ qui (sắp a, b)
```

```
    quicksort(a_head, a_tail);
    quicksort(b_head, b_tail);

    // xây dựng lời giải (nối a đã sắp, m, b đã sắp)
    if(a_tail == NULL)
        head = m_head;
    else
    {
        a_tail->next = m_head;
        head = a_head;
    }
    m_tail->next = b_head;
    if(b_tail == NULL)
        tail = m_tail;
    else
        tail = b_tail;
}
```

- Lưu ý, để hỗ trợ thao tác nối danh sách liên kết hiệu quả, ta dùng thêm con trỏ đến nút cuối của danh sách là `tail`. Cài đặt trên trông hơi dài nhưng nó rất đơn giản và dễ hiểu. Hơn nữa, nó thực hiện *tại chỗ* (in-place) việc sắp xếp (nghĩa là không dời hay cấp phát thêm vùng nhớ mà chỉ thay đổi các liên kết) và cũng đảm bảo tính *ổn định* (stable) (nghĩa là duy trì "thứ tự tương đối trước đó của các phần tử bằng nhau". *Hãy nghiền ngẫm để thấy hết vẻ đẹp của nó!*

Kĩ năng

- Đọc hiểu và viết được các hàm đệ qui trên danh sách liên kết
- Biết cách vận dụng đệ qui để giải các bài toán trên danh sách liên kết

Lưu ý

- Chia-để-trị, thu-để-trị và đặc biệt đệ qui là những phương pháp giải quyết vấn đề, phương pháp tư duy và lập trình rất quan trọng nên sinh viên cần rèn luyện nhiều để quen thuộc và thành thạo

Bài tập

1. Cài đặt lại Quicksort bằng đệ qui trên mảng.
2. Tìm hiểu và cài đặt đệ qui thuật toán Merge sort trên danh sách liên kết.
3. Làm lại các bài tập trên danh sách liên kết bằng kĩ thuật đệ qui.